

[INICIAR SESIÓN](#)[NUESTROS PLANES](#)[TODOS LOS CURSOS](#)[FORMACIONES](#)[CURSOS](#)[PARA EMPRESAS](#)[ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE](#)

# Optimización de hiperparámetros



igor-nascimento-flipe

27 de Marzo



La biblioteca **Scikit Learn**, en el lenguaje de programación **Python**, está orientada al *Machine Learning*, con el objetivo de predecir determinadas acciones a través de patrones en una base de datos. Tiene varias **funciones** que requieren **parámetros** para la perfecta ejecución del código y algunas de ellas asumen un estado por defecto (default), devolviendo un valor que, hasta ese momento, es el mejor resultado que podemos encontrar.

Pero, ¿que son realmente?

De hecho, podemos modificar los parámetros de estas funciones, llamados **hiperparámetros**, para mejorar el resultado. El nombre de este proceso se denomina **ajuste de hiperparámetros**.

En este artículo, utilizaremos una base de datos preparada que proporciona el propio sklearn, para que podamos centrarnos en la optimización de hiperparámetros. Vamos a utilizar la base de datos de [wisconsin de cáncer de mama](#) que contiene información sobre nódulos y los clasifica como benignos o malignos.

## Cargando la base de datos

Primero, carguemos la base de datos.

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

A continuación, convirtamos esta base en un Pandas DataFrame para que los datos sean más fáciles de visualizar.

```
import pandas as pd
df_feature = pd.DataFrame(data=data['data'], columns=data['feature_names'])
df_feature.head()
```



```
1 df_feature = pd.DataFrame(data=data['data'], columns=data['feature_names'])
2 df_feature.head()
```

|   | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | radius error | texture error | perimeter error | area error | smoothness error | compactness error | concavity error | concave points error |
|---|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|---------------------|---------------|------------------------|--------------|---------------|-----------------|------------|------------------|-------------------|-----------------|----------------------|
| 0 | 17.99       | 10.38        | 122.80         | 1001.0    | 0.11840         | 0.27760          | 0.3001         | 0.14710             | 0.2419        | 0.07871                | 1.0950       | 0.9053        | 8.589           | 153.40     | 0.006399         | 0.04904           | 0.05373         | 0.01587              |
| 1 | 20.57       | 17.77        | 132.90         | 1326.0    | 0.08474         | 0.07864          | 0.0869         | 0.07017             | 0.1812        | 0.05667                | 0.5435       | 0.7339        | 3.398           | 74.08      | 0.005225         | 0.01308           | 0.01860         | 0.01340              |
| 2 | 19.69       | 21.25        | 130.00         | 1203.0    | 0.10960         | 0.15990          | 0.1974         | 0.12790             | 0.2069        | 0.05999                | 0.7456       | 0.7869        | 4.585           | 94.03      | 0.006150         | 0.04006           | 0.03832         | 0.02058              |
| 3 | 11.42       | 20.38        | 77.58          | 386.1     | 0.14250         | 0.28390          | 0.2414         | 0.10520             | 0.2597        | 0.09744                | 0.4956       | 1.1560        | 3.445           | 27.23      | 0.009110         | 0.07458           | 0.05661         | 0.01867              |
| 4 | 20.29       | 14.34        | 135.10         | 1297.0    | 0.10030         | 0.13280          | 0.1980         | 0.10430             | 0.1809        | 0.05883                | 0.7572       | 0.7813        | 5.438           | 94.44      | 0.011490         | 0.02461           | 0.05688         | 0.01885              |

Ahora vamos a crear una tabla con una sola columna, llamada **Pandas Series for the Targets**, que son la clasificación de los datos. Usemos el método único para visualizar que solo tendremos dos clases, la 0 y 1.

```
ser_targets = pd.Series(data=data['target'], name='benign')
ser_targets.unique()
```

Ahora transfiramos los datos a las variables X e Y por convención.

```
X = df_feature
y = ser_targets
```

## Primer modelo

Exploremos el modelo **DecisionTreeClassifier**, que es un árbol de decisión que clasifica los datos, en nuestro caso, entre 0 y 1. Primero usamos sus **hiperparámetros** definidos por **defecto**.

```
from sklearn.tree import DecisionTreeClassifier
modelo_arvore = DecisionTreeClassifier()
```

Un punto importante es que vamos a utilizar el concepto de validación cruzada, donde tratamos de minimizar los efectos que la aleatoriedad de los datos seleccionados para el entrenamiento y la prueba pueden tener en nuestro resultado. Lo que hará **cross\_validate** es separar nuestros datos en 5 grupos y usarlos para entrenar varios modelos, en este caso 5 veces porque configuramos **cv=5**, variando qué datos serán para entrenamiento y cuáles para prueba.

```
results = cross_validate(modelo_arvore, X, y, cv=5,
```



El **cross\_validate** devolverá los resultados que contienen la información principal que calculó, centrémonos en **mean\_train\_score** y **mean\_test\_score** que son, respectivamente, la precisión promedio con los valores de entrenamiento y la precisión promedio con los valores de prueba.

```
print(f"mean_train_score {np.mean(results['train_score']):.2f}") print(f"mean_
```



Tenemos:

```
mean_train_score 1.00
```

```
mean_test_score 0.92
```

Tuvimos un buen resultado de Score, pero tuvimos una precisión del 100% en los datos de entrenamiento, lo que indica un sobreajuste (**overfitting**) cuando el modelo está especializado en los datos de entrenamiento.

## Optimización de los hiperparámetros

Los hiperparámetros son características de su modelo que se pueden definir a través de parámetros. Por ejemplo, [DecisionTreeClassifier](#) tiene los parámetros **max\_depth** y **min\_samples\_split** que, según el valor, entregan un modelo que se adapta mejor a sus datos.

Pero eso nos lleva a una pregunta. Basta con mirar la documentación de uno de estos algoritmos para darse cuenta de la infinidad de parámetros que tenemos y, en consecuencia, de muchas posibilidades. Para este problema tenemos dos soluciones: definir los valores e hiperparámetros que vamos a explorar o explorar aleatoriamente. Estas dos estrategias se implementan en scikit-learn, [GridSearchCV](#) y [RandomizedSearchCV](#)

Comencemos con **GridSearchCV**. Nos permite definir un espacio que queremos explorar, por ejemplo:

```
from sklearn.model_selection import GridSearchCV
```

```
espaco_de_parametros = {  
    "max_depth" : [3, 5],  
    "min_samples_split" : [32, 64, 128],  
    "min_samples_leaf" : [32, 64, 128],
```

```
"criterion" : ["gini", "entropy"]  
}
```

Aquí definimos un diccionario donde la clave es el nombre del parámetro que queremos optimizar y el valor es una lista de valores que queremos que explore. Entonces, en este caso, entrena el árbol usando una profundidad máxima, **max\_depth**, 3 y 5.

Para usar **GridSearchCV** repetiremos algunos parámetros que usamos en **cross\_validate**. Lo nuevo es que no vamos a enviar x e y, y vamos a usar el parámetro **param\_grid**, que será nuestro diccionario.

```
modelo_arbol = DecisionTreeClassifier()  
  
clf = GridSearchCV(modelo_arbol, espacio_de_parametros, cv=5, return_train_score=True)  
search = clf.fit(X, y)  
results_GridSearchCV = search.cv_results_  
indice_mejores_parametros = search.best_index_
```

Otra diferencia es que tendremos más de un modelo, por lo que debemos seleccionar el que mejor se desempeñó. Accedemos a **search.bestindex** y luego lo seleccionamos dentro de nuestros resultados.

```
print(f"mean_train_score {results_GridSearchCV['mean_train_score'][indice_mejores_parametros]}
```

Resultado:

```
mean_train_score 0.94
```

```
mean_test_score 0.92
```

Para recuperar los parámetros que tuvieron el mejor rendimiento, usemos la llave **params**:

```
results_GridSearchCV['params'][indice_melhores_parametros]
```

Resultado:

```
{'criterion': 'gini',  
 'max_depth': 3,  
 'min_samples_leaf': 32,  
 'min_samples_split': 32}
```

Con esto, logramos descubrir el mejor modelo entre el grupo de parámetros que definimos, pero esto puede ser un problema, porque el espacio de posibilidades es mucho más grande. Entonces, la mejor combinación de parámetros podría estar en otro lugar de ese espacio. Con eso en mente, tenemos la segunda solución, **RandomizedSearchCV**. En él definimos el espacio que queremos explorar y probará aleatoriamente las combinaciones de hiperparámetros, además de probar la cantidad que determinará el parámetro **n\_iter**, que es el número de interacciones.

```
from sklearn.model_selection import RandomizedSearchCV
```


Primero definamos este espacio. Tenga en cuenta que ahora podemos incluir una gran cantidad de posibilidades, ya que no todas estarán probadas. Por ejemplo, en **min\_samples\_split** probamos números aleatorios entre 32 y 129.

```
from scipy.stats import randint  
  
espacio_de_parametros = {  
    "max_depth" : randint(1, 10),  
    "min_samples_split" : randint(32, 129),  
    "min_samples_leaf" : randint(32, 129),  
    "criterion" : ["gini", "entropy"]  
}
```

Con el espacio definido, podemos pasar a usar **RandomizedSearchCV**. Fíjate que lo vamos a utilizar de forma similar a **GridSearchCV**, pero con un parámetro más, **n\_iter**, que es el número de combinaciones que intentará.


```
modelo_arbol = DecisionTreeClassifier()

clf = RandomizedSearchCV(modelo_arbol, espacio_de_parametros, random_state=SEE
search = clf.fit(X, y)
results_RandomizedSearchCV = search.cv_results_
indice_mejores_parametros = search.best_index_
```



Para poder comparar los resultados, modifiqué la función presentada anteriormente para usar estos optimizadores de hiperparámetros:

```
print(f"mean_train_score {results_RandomizedSearchCV['mean_train_score'][indice_
print(f"mean_test_score {results_RandomizedSearchCV['mean_test_score'][indice_
```



Resultado:

```
mean_train_score 0.92
mean_test_score 0.91
```

Para recuperar los parámetros que tuvieron el mejor rendimiento, usemos la llave params:

```
results_RandomizedSearchCV['params'][indice_melhores_parametros]
```

Resultado:

```
{'criterion': 'gini',
 'max_depth': 4,
 'min_samples_leaf': 34,
 'min_samples_split': 105}
```

Con esto podemos ver que podemos explorar más los algoritmos. Recomendando explorar la documentación de los algoritmos con los que está trabajando. Busque hiperparámetros para explorar y encontrar los mejores para sus datos y propósito.

En este [notebook](https://www.aluracursos.com/blog/optimizacion-de-hiperparametros?utm_source=gnarus&utm_medium=timeline) encontrarás el código completo del proyecto.



## Igor Nascimento Alves

Soy licenciado en Informática. Trabajo como instructor de Data Science y Machine Learning en Grupo Alura, teniendo como principales intereses en tecnología: creación de modelos y análisis de datos. En mi tiempo libre veo y analizo datos de baloncesto y me encanta escuchar podcasts de humor como Nerdcast y Jujubacast.

Cursos de Data Science

ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE

**En Alura encontrarás variados cursos sobre Data Science.  
¡Comienza ahora!**

**SEMESTRAL**

**US\$49,90**

un solo pago de US\$49,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español



- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 6 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**

[Paga en moneda local en los siguientes países](#)

**ANUAL**

**US\$79,90**

un solo pago de US\$79,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓

Estudia las 24 horas, los 7 días de la semana

✓ Foro y comunidad exclusiva para resolver tus dudas

✓ Acceso a todo el contenido de la plataforma por 12 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**

[Paga en moneda local en los siguientes países](#)

Acceso a todos  
los cursos

Estudia las 24 horas,  
dónde y cuándo quieras

Nuevos cursos  
cada semana

## NAVEGACIÓN

PLANES  
INSTRUCTORES  
BLOG  
POLÍTICA DE PRIVACIDAD  
TÉRMINOS DE USO  
SOBRE NOSOTROS  
PREGUNTAS FRECUENTES

## ¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

## BLOG

PROGRAMACIÓN  
FRONT END  
DATA SCIENCE  
INNOVACIÓN Y GESTIÓN  
DEVOPS

AOVS Sistemas de Informática S.A  
CNPJ 05.555.382/0001-33

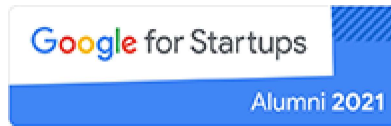
## SÍGUENOS EN NUESTRAS REDES SOCIALES



## ALIADOS



En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

## CURSOS

### Cursos de Programación

Lógica de Programación | Java

### Cursos de Front End

HTML y CSS | JavaScript | React

### Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

### Cursos de DevOps

Docker | Linux

### Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento